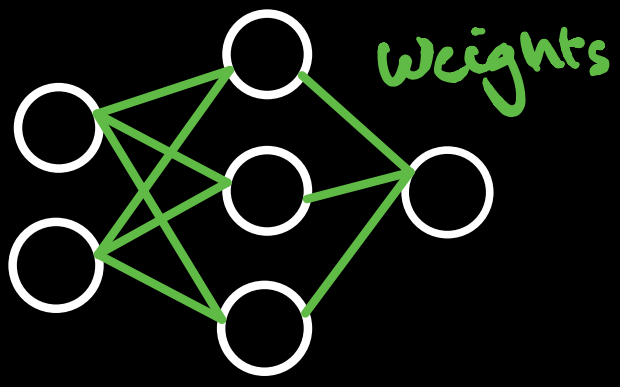
A neural network

in plain Numpy

1. initialize network



initiate with small numbers

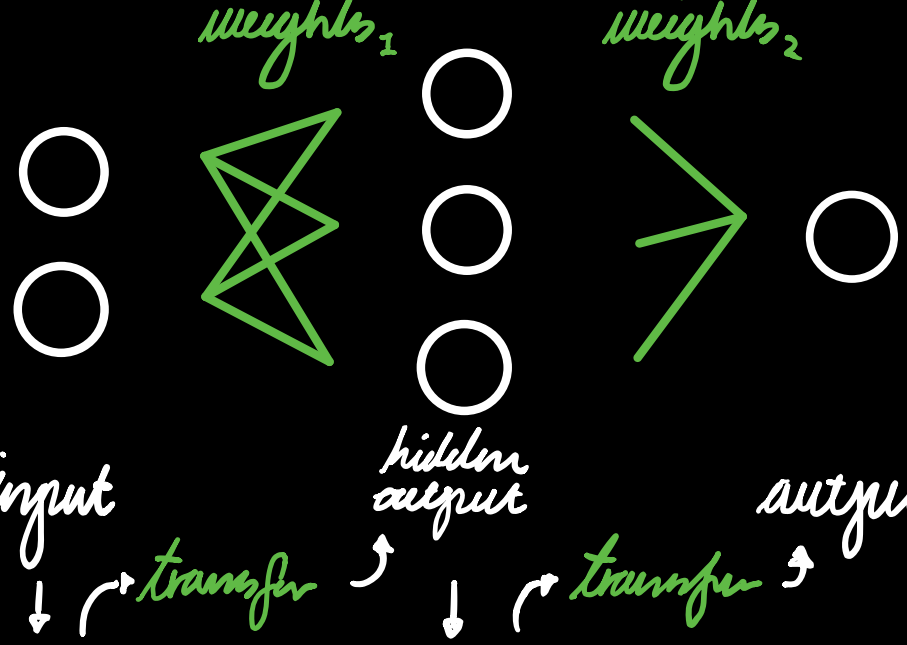
2. the forward propagate

activation → transfer → forward prop

activation : $\sum_{i=1}^{N} w_i \cdot input_i + bias$

transfer : $\frac{1}{1 + e^{-activation}}$

forward prop :



activation activation

3. back propagate error

transfer derivation → error backprop

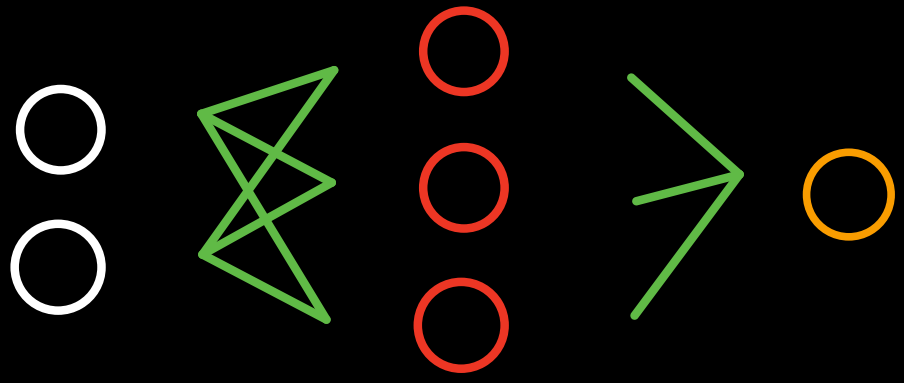
transfer derivation: $output \times (1 - output)$

(derivative of sigmoid)

error backprop:

$error_{OUTPUT} = (expected - output) \cdot transfer\ derivation$

where expected is the expected output from each neuron.



$error_{HIDDEN} = (weight_k \cdot error_j) \cdot transfer\ derivation$

where $error_j$ is the error from neuron j and $weight_k$ is the weight connecting the k'th neuron to the current neuron.

4. train network

update weights → train network

update: weight + learning rate · error · input

train:

```
for epoch in epochs:
    sum(error) = 0
    for row in train:
        1. forward prop
        2. calculate expected
        3. backprop error
        4. update weights
```

5. predict

forward prop → argmax

forward: take trained network and forward prop the new input.

argmax: given the new output, choose the highest input with argmax